Muhammed Rıdvan Kaya ORCİD: 0009-0008-9254-6532

Kaya Takvimi ve Bağımlı İki Örneklemde Dağılım Kayması Testi: Astronomik Bir Zaman Çerçevesinde İstatistiksel Analiz

Giriş

Zamanı ölçmek, insanlığın en eski ve en köklü zihinsel faaliyetlerinden biridir. Gregoryen takvim, idari ve ticari hayatı düzenlemekte son derece işlevseldir; ancak doğanın döngüsel ritmiyle —mevsimler, güneş ve ay hareketleri— organik bir bağ kurmaktan uzak, tamamen sözleşmeye dayalı bir sistemdir. Burada incelenen Kaya Takvimi, tam da bu boşluğu doldurmak amacıyla tasarlanmış, Güneş'in ekliptik boylamını temel referans noktası olarak kullanan astronomik tabanlı bir takvim sistemidir. Kaya Takvimi'nin temel hesaplama altyapısı NASA'nın JPL (Jet Propulsion Laboratory) gezegensel efemerisine —de440.bsp dosyasına— ve Python'un skyfield kütüphanesine dayanmaktadır; bu sayede takvim, hesaplamalı bir soyutlamadan öte, gerçek gök cismi konumlarını milisaniye düzeyinde takip eden hassas bir astronomik araça dönüşmektedir.

Bu çalışma iki ana eksende ilerlemektedir. Birinci eksende Kaya Takvimi'nin matematiksel ve astronomik temelleri ayrıntılı biçimde ele alınmaktadır. İkinci eksende ise bu takvim çerçevesinin bir istatistiksel analiz platformuna nasıl dönüştürülebileceği gösterilmektedir: Kaya aylarına göre gruplandırılmış zaman serileri arasında dağılım kayması olup olmadığı, bağımlı örnekleme uygun olan blok permütasyon testi ve blok bootstrap yöntemleriyle sınanmaktadır. Söz konusu metodolojik yaklaşım, takvim araştırmasını aşarak iklim bilimi, ekoloji ve ekonometri gibi alanlarda doğrudan kullanılabilir niteliktedir.

Kaya Takvimi'nin Astronomik Temelleri

Tropik Yıl ve Ekinoks Referansı

Kaya Takvimi'nin temel zaman birimi tropik yıldır. Tropik yıl, Güneş'in ortalama yay hızıyla ölçüldüğünde bahar ekinoksundan bahar ekinoksuna geçen süredir ve yaklaşık 365,2421896698 gün sürer. Bu değer Gregoryen takvimin temel aldığı 365,2425 günlük ortalamadan ufak ama istatistiksel açıdan anlamlı biçimde farklıdır; uzun vadede bu fark birikimli bir kaymaya yol açar.

Referans başlangıç noktası (epoch), 20 Mart 2000 tarihindeki bahar ekinoksudur; daha kesin ifadeyle, Güneş'in ekliptik boylamının tam olarak 0° değerini geçtiği an. Bu an UTC 07:35 olarak sabitlenmiştir. Epoch seçimi keyfi değildir: bahar ekinoksu, hem astronomik hem kültürel açıdan evrensel bir yenilenme noktasını simgeler ve tropikal yıl tanımıyla doğrudan örtüşür.

Kaya Takvimi'nin sıfır noktasından (epoch) itibaren geçen süre şu formülle hesaplanmaktadır:

gün_sayısı = (hedef_an_UTC - epoch_UTC) / 86400 saniye

Sivil günün başlangıcı gece yarısı değil saat 07:35 olarak tanımlanmıştır; bu tercih, astronomik günün —yani ekinoks referansının— sivil yaşamla uyumlu biçimde hizalanmasını sağlar. Yazılımsal olarak bu durum, ham gün sayısına 7,5833.../24

oranında bir kaydırma eklenerek yönetilmektedir.

Güneş Boylamına Dayalı Ay Sistemi

Kaya Takvimi'nde aylar, Gregoryen takvimde olduğu gibi Ay'ın hareketi ya da sözleşmesel gün sayılarıyla değil, Güneş'in ekliptik boylamı esas alınarak tanımlanmaktadır. Güneş 360°'lik ekliptiği yılda bir kez dolaşır; bu yay 30°'lik 12 eşit dilime bölündüğünde 12 ay elde edilir:

| Boylam Aralığı | Ay Adı | Karşılık Geldiği Mevsimsel Dönem |

|---|---|---|

| 0° – 30° | Filiz | İlkbahar başı |

| 30° – 60° | Çiçek | İlkbahar ortası |

| 60° – 90° | Yeşeren | İlkbahar sonu |

| 90° – 120° | Güneş | Yaz başı |

| 120° – 150° | Köz | Yaz ortası |

| 150° – 180° | Hasat | Yaz sonu |

| 180° – 210° | Rüzgar | Sonbahar başı |

| 210° – 240° | Sarı | Sonbahar ortası |

| 240° – 270° | Yağmur | Sonbahar sonu |

| 270° – 300° | Ayaz | Kış başı |

| 300° – 330° | Kar | Kış ortası |

| 330° – 360° | Uykuda | Kış sonu |

Bu sistemin Gregoryen takvime kıyasla en belirgin üstünlüğü, her ayın gerçek bir mevsimsel dilime karşılık gelmesidir. Güneş tam olarak 120°'de iken her zaman astronomik yaz başlangıcındayız demektir; bu değer sabit ve kesindir. Gregoryen takvimde ise yaz gündönümü 20 ile 22 Haziran arasında değişebilir.

NASA JPL Efemerisinden Gerçek Zamanlı Boylam Hesabı

Güneş ve Ay'ın ekliptik boylamları skyfield kütüphanesi aracılığıyla JPL'nin de440.bsp efemerisinden doğrudan alınmaktadır. Bu işlem, Dünya merkezli gözlemcinin bakış açısından görünen (apparent) konumu hesaplar; yani sapma (aberration) ve ışık zaman gecikmesi (light travel time) düzeltmeleri otomatik olarak uygulanmış olur. Kod düzeyinde işlem şu adımları izler: önce zaman damgası skyfield zaman nesnesine dönüştürülür, ardından Dünya'dan Güneş'e ve Ay'a gözlemsel bağlantı kurulur, ekliptik çerçeveye (ecliptic_frame) dönüştürme yapılır ve boylamın derece cinsinden değeri mod 360 operasyonuyla 0°–360° aralığına normalize edilir. Hesaplanan bu değerler tekrarlı API çağrılarının maliyetini düşürmek amacıyla bir

sözlükte önbelleğe alınmaktadır.

Ay Evresi Hesabı

Ay'ın evresi, Ay ile Güneş'in ekliptik boylamları arasındaki açısal farkla (sinodik açı) belirlenir:

faz_açısı = (Ay_boylamı − Güneş_boylamı) mod 360°

Bu açı sekiz dilime bölünerek sekiz Ay evresine —yeniaydadn son hilale— karşılık getirilir. Yeniay 0° (veya 360°) civarında, dolunay ise 180° civarında gerçekleşir. Hesaplama saf geometriktir ve takvim sözleşmelerine bağlı değildir.

6 Günlük Hafta

Kaya Takvimi 7 günlük yerine 6 günlük bir hafta kullanmaktadır. Haftanın günleri Gün-A'dan Gün-F'ye kadar harfsiz bir nitelendirmeyle tanımlanmıştır; bu sayede kültürel yüklü isimlerden kaçınılmaktadır. Epoch'tan itibaren geçen tam gün sayısının 6'ya bölümünden kalan, o günün hafta içindeki yerini belirler. 6'nın seçimi yalnızca kültürel bir tercih değil, matematiksel bir özelliktir: 6, 12 ile tam bölünebilir olduğundan aylık döngülerle daha düzgün örtüşür ve karma matematiksel modellerde faydalı bir indeksleme sağlar.

İstatistiksel Çerçeve: Bağımlı İki Örneklemde Dağılım Kayması Testi

Problemi Tanımlamak

Doğal veriler nadiren bağımsız ve özdeş dağılımlıdır (i.i.d.). Hava sıcaklıkları, biyolojik ölçümler, finansal getiriler ve ekolojik gözlemler sistematik olarak zaman içinde ardışık bağımlılık gösterir: dünkü değer, bugünkü değeri etkiler. Bu durum klasik parametrik testlerin —Student t-testi, Mann-Whitney U testi gibi— varsayımlarını ihlal eder; bağımsızlık varsayımı sağlanmadığında bu testlerin I. tip hata oranları —yanlış pozitif oranı— nominal düzeyi aşar.

Kaya Takvimi çerçevesinde ise soru şudur: Belirli Kaya aylarında gözlemlenen bir değişken, diğer aylarda gözlemlenenlerden istatistiksel olarak farklı bir dağılıma mı sahiptir? Bu soru mevsimsellik araştırmalarının, fenoloik çalışmaların ve hava-iklim analizlerinin doğal bir uzantısıdır. Ancak aylar arasındaki gözlemler bağımsız değil; bir AR(1) süreci ya da daha genel zayıf bağımlı bir sürecin parçasıdır.

Temel Varsayım: Zayıf Bağımlılık (α-Karıştırma)

İncelenen metodolojinin geçerliliği için gerekli temel varsayım, serinin **α-karıştırıcı** (alpha-mixing) bir süreç olduğudur. Kaba ifadeyle bu, geçmişin uzak geleceği sıfıra yakınsayan bir hızda etkilediği anlamına gelir: yeterince uzak iki zaman noktasındaki gözlemler neredeyse bağımsız davranır. Durağan AR(1) süreçleri ve pek çok ARMA modeli bu koşulu sağlar. Önemli bir uyarı: yazılımda kullanılan **Genişletilmiş Dickey-Fuller (ADF) testi** durağanlığı sınarken α-karıştırma koşulunu kanıtlamaz; bu koşul teorik bir kabulden ibarettir ve pratik uygulamalarda dikkatli biçimde değerlendirilmelidir.

Kolmogorov-Smirnov Uzaklığı

İki örneklem için KS istatistiği, ampirik kümülatif dağılım fonksiyonları (ECDF) arasındaki en büyük mutlak fark olarak tanımlanır:

$D_{KS} = \sup_t |F_n(t) - G_m(t)|$

Bu istatistik yüksek derecede yoruma açıktır; dağılımlar arasındaki en büyük sapmanın nerede gerçekleştiğini gösterir. Ancak ağır kuyruklu dağılımlarda merkezi farklara duyarsız kalabilir. Klasik KS testinin p-değerleri bağımlı veriler için hatalıdır; bu nedenle burada blok permütasyon yöntemiyle hesaplanmaktadır.

Wasserstein Uzaklığı (Taşıma Maliyeti)

Wasserstein-1 uzaklığı, diğer adıyla Earth Mover's Distance, bir dağılımı diğerine dönüştürmek için gereken minimum toplam taşıma maliyetidir:

$W_1(F, G) = \int |F(t) - G(t)| \, dt = \int_0^1 |F^{-1}(\tau) - G^{-1}(\tau)| \, d\tau$

İkinci ifade, kantil dönüşümü aracılığıyla hesaplamayı son derece verimli kılar: scipy'nin `wasserstein_distance` fonksiyonu bu formülü kullanmaktadır. Wasserstein uzaklığı KS istatistiğinin aksine dağılımlar arasındaki küresel geometrik farkı yakalar; küçük ama yaygın kaymalar için daha duyarlıdır.

Cliff's Delta: Olasılıksal Üstünlük

Cliff's Delta, x'in y'den büyük olma olasılığı ile x'in y'den küçük olma olasılığının farkıdır:

$\delta = P(X > Y) - P(X < Y)$

Bu istatistik -1 ile +1 arasında değer alır; 0, stokastik eşitliği ifade eder. Cliff's Delta parametrik olmayan ve dağılım varsayımı gerektirmeyen bir etki büyüklüğü ölçüsüdür; test anlamlı olduğunda bile pratik önemi değerlendirmek için kritik bir araçtır. Güven aralığı da blok bootstrap ile hesaplanmaktadır.

Kantil Farkı Fonksiyonu

Dağılım farkını tek bir sayıya indirgemek yerine, tüm kantil spektrumunda nasıl değiştiğini görmek çoğu zaman daha bilgilendiricidir. Kantil farkı fonksiyonu:

$\Delta(\tau) = q_\tau^x - q_\tau^y, \tau \in (0, 1)$

biçiminde tanımlanır. Bu fonksiyon bir grafik üzerinde çizildiğinde şu soruları yanıtlar: Fark dağılımın tamamına mı yayılmış yoksa yalnızca uç değerlerde mi belirgin? Alt ve üst kuyruklar farklı biçimde mi davranıyor? Fark istatistiksel olarak anlamlı mı, yoksa güven bandı sıfırı kapsıyor mu?

Yöntem: Blok Permütasyon Testi

Temel Fikir

Klasik permütasyon testleri gözlemleri rastgele yeniden düzenler ve boş hipotez

altındaki test istatistiğinin dağılımını ampirik olarak oluşturur. Ancak bağımlı veriler için bireysel gözlemleri karıştırmak bağımlılık yapısını bozar ve yanıltıcı sonuçlar üretir. Blok permütasyon testinin çözümü, bireysel gözlemler yerine ardışık blokların yer değiştirmesidir. Blok içi bağımlılık korunur; yalnızca blokların x ve y grubu arasındaki ataması değiştirilir.

Algoritma

Uzunluğu n olan x ve y serileri, her biri blok_boy uzunluğunda n_blok bloğa bölünür. Her permütasyon adımında her blok için bağımsız bir yazı-tura çekilir: yazı gelirse o blok x'te kalır ve karşılık gelen y bloğu y'de; tura gelirse bloklar yer değiştirir. Ardından karıştırılmış seriler üzerinde seçilen test istatistiği —KS, Wasserstein veya başka bir ölçüt— hesaplanır. Bu işlem n_perm kez tekrarlandıktan sonra gözlemlenen istatistiğin permütasyon dağılımının kaç yüzdesilik dilimine düştüğü p-değerini verir.

Kritik bir uygulama ayrıntısı, blokların permütasyondan önce düzleştirilmesidir (.ravel() operasyonu). Numpy'ın concatenate ve reshape operasyonlarının satır-öncelikli sıralama (C-order) varsayımı nedeniyle bu adım atlandığında veri organizasyonu bozulmakta ve test istatistiği hatalı değerler üretmektedir.

Blok Boyutu Seçimi

Blok boyutu kritik bir ayar parametresidir: çok küçük seçilirse bağımlılık yapısı bozulur, çok büyük seçilirse permütasyonların etkin sayısı azalır ve testin gücü düşer. Kullanılan sezgisel kural, Hall (1985) ve sonraki çalışmalardan ilham alan n^(1/3) kuralıdır. Optimal blok boyutu gerçekte serinin otokorelasyon yapısına bağlıdır; ampirik seçim için Politis ve Romano'nun (1994) bant genişliği seçim algoritmaları da kullanılabilir. Duyarlılık analizi, birden fazla blok boyutu ve permütasyon sayısı kombinasyonu için sonuçları raporlayarak bu belirsizliği şeffaf biçimde ele almaktadır.

Blok Bootstrap ile Güven Aralıkları

Hareketli Blok Bootstrap

Blok bootstrap, Künsch (1989) tarafından önerilen ve zaman serilerinde bağımlılığı koruyan bir yeniden örnekleme yöntemidir. Hareketli blok bootstrap versiyonunda başlangıç noktaları 1 ile n - blok_boy + 1 arasından rastgele ve değiştirmeli olarak seçilir; bu sayede bloğun sınırları verinin herhangi bir noktasında başlayabilir. Seçilen bloklar uçuca eklenerek n uzunluğunda yeni bir seri oluşturulur.

Kantil Farkı Güven Bandı

Bootstrap prosedürü her iterasyonda x ve y'yi bağımsız olarak yeniden örnekler ve bu yeniden örneklenmiş seriler üzerinde kantil farkı fonksiyonu hesaplar. Bin iterasyon sonunda her τ değeri için bir bootstrap dağılımı elde edilir. Bunun alt ve üst yüzdelik dilimleri —örneğin %2,5 ve %97,5— güven bandını oluşturur. Güven bandının tüm τ boyunca sıfırı kapsadığı bölgeler, dağılımlar arasında anlamlı fark olmayan kantilleri işaret eder.

Uygulama: Kaya Ayları Arası Dağılım Karşılaştırması

Simülasyon Tasarımı

Metodoloji, Kaya takvimi üzerinde gerçekçi bir uygulama senaryosuyla somutlaştırılmaktadır. 2024 yılının her günü için AR(1) bağımlı bir süreç üretilmektedir:

$$\varepsilon_t = 0.6 \times \varepsilon_{t-1} + \eta_t, \quad \eta_t \sim N(0, 1)$$

Bu süreç orta düzeyde otokorelasyon sergiler; meteorolojik zaman serilerini, biyolojik ölçümleri ve çok sayıda çevresel değişkeni temsil edebilecek genel bir modeldir. Her günün Kaya takvimindeki ayı, o günün Güneş boylamından hesaplanır. Ardından iki Kaya ayı —Güneş (90°–120°, yaz başlangıcı) ve Köz (120°–150°, yaz ortası)— seçilerek bu aylara ait gözlemler ayrılır ve bağımlı iki örneklem karşılaştırması yapılır.

Yorumlama Kılavuzu

Test sonuçları okunurken üç katmanlı bir değerlendirme yapılmalıdır.

Birinci katman — İstatistiksel anlamlılık: KS ve Wasserstein p-değerleri belirlenen alfa düzeyinin (genellikle 0,05) altındaysa boş hipotez —dağılımlar aynıdır— reddedilir. Ancak bu yalnızca "bir fark var" der; farkın büyüklüğü ve yönü hakkında bilgi vermez.

İkinci katman — Etki büyüklüğü: Cliff's Delta, farkın pratik önemini ölçer. Genel yönergeler şu şekildedir: $|\delta| < 0{,}147$ ihmal edilebilir, 0,147–0,330 küçük, 0,330–0,474 orta, 0,474 ve üzeri büyük etki olarak sınıflandırılır. Güven aralığının sıfırı kapsayıp kapsamadığı da dikkate alınmalıdır.

Üçüncü katman — Kantil farkı grafiği: Bu grafik farkın dağılım boyunca nerede yoğunlaştığını gösterir. $\Delta(\tau)$ fonksiyonunun yalnızca yüksek τ değerlerinde —yani dağılımın üst kuyruğunda— güven bandından çıkması, farkın aşırı değerlerde belirginleştiğini ve merkezi eğilim ölçütlerini kullanan testlerin bunu kaçırabileceğini ortaya koyar.

Metodolojik Sınırlılıklar ve Uyarılar

Bu çalışmada uygulanan metodoloji güçlü teorik temellere sahip olmakla birlikte dört temel sınırlılığı şeffaf biçimde kabul etmektedir.

Birinci sınırlılık — Zayıf bağımlılık varsayımı yalnızca kabul edilmektedir: ADF testi durağanlığı sınar; durağanlık α-karıştırma için gerekli ama yeterli değildir. Uzun hafızalı (long-memory) süreçler, yapısal kırılmalar veya periyodik mevsimsellik içeren seriler bu testi geçse bile zayıf bağımlılık koşulunu ihlal edebilir. Araştırmacıların otokorelasyon fonksiyonunu ve kısmi otokorelasyon fonksiyonunu incelemesi önerilir.

İkinci sınırlılık — Test asimptotik olarak geçerlidir: Blok permütasyon testinin teorik garantisi büyük örneklemler için geçerlidir; küçük aylık dilimler —ki bunlar genellikle 28–31 gözlemden ibarettir— için I. tip hata oranı nominal düzeyden sapabilir. Gerçek hata oranını değerlendirmek için simülasyon çalışmaları yapılabilir.

Üçüncü sınırlılık — Blok boyutu seçimi kesin değildir: n^(1/3) kuralı makul bir başlangıç noktasıdır, ancak optimal değildir. Yanlış blok boyutu II. tip hatayı —gerçek

bir farkı gözden kaçırmayı— artırır. Duyarlılık analizinin sunulması bu belirsizliği hafifletir ama ortadan kaldırmaz.

Dördüncü sınırlılık — Nedensellik değil ilişki: Test yalnızca dağılımların farklı olduğunu söyler; bu farklılığın astronomik konumdan mı, karıştırıcı değişkenlerden mi yoksa başka bir etkenlerden mi kaynaklandığını açıklamaz. Güneş boylamıyla ilişkili bir fark bulmak, Güneş boylamının o değişkeni etkilediği anlamına gelmez.

Daha Geniş Bir Perspektif: Astronomik Zaman Çerçevelerinin İstatistiksel Değeri

Kaya Takvimi'nin yalnızca estetik veya kültürel bir proje olmadığı, bu çalışmayla açıkça ortaya konmaktadır. Güneş boylamı temelli ay tanımı, Gregoryen takvimin yapay sınırlarını aşan doğal bir mevsimsel gruplama sağlar. Bu gruplama, fenoloji —bitki ve hayvan yaşam döngülerinin mevsimlere yanıtını inceleyen bilim dalı— araştırmalarında, tarımsal verimlilik analizlerinde ve iklim değişikliği çalışmalarında güçlü bir metodolojik araç olabilir. Güneş boylamı, mevsimleri takvim aylarından çok daha kesin ve tutarlı biçimde temsil eder; çünkü mevsimlerin fiziği yeryüzünün Güneş etrafındaki konumundan kaynaklanmaktadır ve Kaya Takvimi bu konumu doğrudan kullanmaktadır.

Öte yandan astronomik zaman çerçevelerinin istatistiksel analizle birleştirilmesi tarihte nadir görülen bir yaklaşımdır. Oysa modern hesaplamalı araçlar —skyfield, scipy, statsmodels— bu yaklaşımı günümüzde son derece erişilebilir kılmaktadır. Bu çalışmanın sunduğu altyapı, iklim istasyonu verilerini Kaya aylarına göre gruplandırarak mevsimsel iklim değişkenliğini analiz etmek, fenolojik gözlemleri astronomik döngülerle ilişkilendirmek veya tarımsal getirilerin güneş boylamıyla nasıl değiştiğini incelemek için doğrudan kullanılabilir.

Sonuç

Burada, iki görünürde bağımsız alanı —astronomik takvim tasarımı ve bağımlı örneklem istatistiği— tek bir tutarlı çerçevede birleştirmiştik. Kaya Takvimi, NASA JPL efemerisi üzerine inşa edilmiş, Güneş'in gerçek ekliptik konumunu ay tanımının temeline alan ve mevsimsel döngülerle organik bir uyum içinde olan astronomik bir zaman sistemidir. Blok permütasyon testi, blok bootstrap ve kantil farkı analizi ise bu takvim çerçevesini istatistiksel bir araştırma platformuna dönüştüren metodolojik omurgayı oluşturmaktadır.

Çalışmanın temel metodolojik katkısı şudur: bağımlı zaman serilerini takvim sisteminin doğal birimleri —astronomik olarak tanımlanan aylar— üzerinden gruplandırmak ve bu gruplama sonucu elde edilen örneklemler arasındaki dağılım farkını bağımlılığa duyarlı yöntemlerle sınamak mümkündür. Bu yaklaşım hem istatistiksel açıdan sağlamdır hem de mevsimsel örüntülerle ilgilenen her bilim dalı için uyarlanabilir niteliktedir.

Gelecek çalışmalar için birkaç öneri sunulabilir: optimal blok boyutu seçimi için Politis -Romano algoritmasının entegrasyonu; çoklu Kaya ayı karşılaştırmaları için çoklu test düzeltmelerinin (Bonferroni, Benjamini-Hochberg) uygulanması; gerçek meteorolojik ya da fenolojik veri kümeleri üzerinde doğrulama; ve son olarak astronomik zaman

çerçevesinin döngüsel bileşenlerini —Ay evresi, mevsimsel eğilim— birlikte modelleyen çok değişkenli uzantılar. Bu adımların her biri, astronomik zaman çerçeveleri ile bağımlı veri analizinin kesişim noktasında yeni bir araştırma alanının kapılarını aralamaktadır.

Simülasyon

```
# ================================================================================

# 📦 GEREKLİ KÜTÜPHANELER (ilk çalıştırmada pip ile yüklenmeli)

# ================================================================================

# !pip install skyfield numpy scipy statsmodels matplotlib

import math

import numpy as np

from scipy.stats import ks_2samp, wasserstein_distance

from statsmodels.tsa.stattools import adfuller

from typing import Callable, Optional, Tuple, List, Dict, Any

import warnings

from datetime import datetime, timezone, timedelta

from skyfield.api import load

from skyfield.framelib import ecliptic_frame

# Görselleştirme için matplotlib

import matplotlib.pyplot as plt

# ================================================================================

# 🌍 TÜRKİYE SAATİ ve SKYFIELD (NASA)

# ================================================================================

TR = timezone(timedelta(hours=3))
```

```
ts = load.timescale()
# İlk kullanımda de440.bsp dosyasını indirir (yaklaşık 1.2 GB)
try:
    eph = load('de440.bsp')
except FileNotFoundError:
    print("de440.bsp bulunamadı. Skyfield otomatik indirecek...")
    eph = load('de440.bsp')
earth = eph['earth']
sun = eph['sun']
moon = eph['moon']
#
=================================================================
=========
# 🌍 EPOCH (Ekinoks referansı, 2000 yılı bahar ekinoksu)
#
=================================================================
=========
EPOCH = datetime(2000, 3, 20, 7, 35, 0, tzinfo=timezone.utc)
BASE_TROPICAL_YEAR = 365.2421896698
#
=================================================================
=========
# 🌿 KAYA AYLARI
#
=================================================================
=========
KAYA_MONTHS = [
    "Filiz", "Çiçek", "Yeşeren",
    "Güneş", "Köz", "Hasat",
    "Rüzgar", "Sarı", "Yağmur",
    "Ayaz", "Kar", "Uykuda"
```

```
]

# ================================================================================

# 📆 6 GÜNLÜK HAFTA

# ================================================================================

WEEK = ["Gün-A", "Gün-B", "Gün-C", "Gün-D", "Gün-E", "Gün-F"]


def six_day_week(day: int) -> str:

    return WEEK[day % 6]

# ================================================================================

# 🌞🌙 GERÇEK BOYLAMLAR (NASA JPL) – Önbellek ile tekrarlı çağrıları engelle

# ================================================================================

_longitude_cache = {}

def get_longitudes(dt: datetime) -> Tuple[float, float]:

    """Güneş ve Ay'ın ekliptik boylamlarını (derece) döndürür. Sonuçları önbelleğe alır."""

    dt_key = dt.astimezone(timezone.utc).replace(microsecond=0)

    if dt_key in _longitude_cache:

        return _longitude_cache[dt_key]

    t = ts.from_datetime(dt_key)

    e = earth.at(t)

    # Güneş

    sun_pos = e.observe(sun).apparent().frame_latlon(ecliptic_frame)

    Ls = sun_pos[1].degrees % 360

    # Ay
```

```
    moon_pos = e.observe(moon).apparent().frame_latlon(ecliptic_frame)
    Lm = moon_pos[1].degrees % 360
    _longitude_cache[dt_key] = (Ls, Lm)
    return Ls, Lm
def moon_phase(dt: datetime) -> str:
    """Ay evresini (Türkçe isim ve emoji) döndürür."""
    Ls, Lm = get_longitudes(dt)
    phase = (Lm - Ls) % 360
    if phase < 22.5 or phase >= 337.5:
        return "🌑 Yeniay"
    elif phase < 67.5:
        return "🌒 Hilal"
    elif phase < 112.5:
        return "🌓 İlkdördün"
    elif phase < 157.5:
        return "🌔 Şişkin Ay"
    elif phase < 202.5:
        return "🌕 Dolunay"
    elif phase < 247.5:
        return "🌖 Küçülen Ay"
    elif phase < 292.5:
        return "🌗 Sondördün"
    else:
        return "🌘 Son Hilal"
#
==========================================================================
=========
# ⏳ GÜN HESABI (EPOCH'tan itibaren gün sayısı)
```

```
#
==========================================================================
=========
def to_days(dt: datetime) -> float:
    dt_utc = dt.astimezone(timezone.utc)
    return (dt_utc - EPOCH).total_seconds() / 86400.0
def get_civil_components(dt: datetime) -> Dict[str, Any]:
    """
    Sivil takvim bileşenleri (Kaya takvimi için).
    Sivil gün başlangıcı: 07:35 (yerel saatte) astronomik günü takip eder.
    """
    days = to_days(dt)
    # Sivil kaydırma: 07:35 → gece yarısı (07:35/24 = 0.316)
    civil_days = days + (7.5833333 / 24.0)
    year = int(days / BASE_TROPICAL_YEAR)
    day_int = int(civil_days % BASE_TROPICAL_YEAR)
    return {
        "year": year,
        "day": day_int + 1,
        "week": six_day_week(day_int),
        "time": dt.astimezone(TR).strftime("%H:%M")
    }
def gregorian_to_kaya(dt: datetime) -> Dict[str, Any]:
    """Gregoryen tarihi Kaya takvimi formatına dönüştürür."""
    Ls, Lm = get_longitudes(dt)
    moon = moon_phase(dt)
    c = get_civil_components(dt)
    month = KAYA_MONTHS[int(Ls // 30)]
    return {
```

```
        "year": c["year"],
        "month": month,
        "day": c["day"],
        "weekday": c["week"],
        "moon_phase": moon,
        "solar_longitude": round(Ls, 6),
        "lunar_longitude": round(Lm, 6),
        "formatted": (
            f"Kaya Takvimi: {c['day']} {month} ({c['week']}) {c['year']} | "
            f"{moon} | ☀️ {Ls:.3f}° | 🌙 {Lm:.3f}° | Saat {c['time']}"
        )
    }
def get_full_date(dt: datetime) -> str:
    return gregorian_to_kaya(dt)["formatted"]
#
==========================================================================
=========
# 📈 İSTATİSTİKSEL TEST ÇERÇEVESİ (bağımlı iki örneklem, blok tabanlı)
#
==========================================================================
=========
def adf_duraganlik_testi(seri: np.ndarray, alfa: float = 0.05) -> bool:
    """Genişletilmiş Dickey-Fuller testi ile durağanlık kontrolü."""
    sonuc = adfuller(seri, autolag='AIC')
    return sonuc[1] < alfa
def amp_dag_fonk(x: np.ndarray) -> Callable:
    """Ampirik yığmalı dağılım fonksiyonu."""
    x_sirali = np.sort(x)
    def f(t):
        return np.searchsorted(x_sirali, t, side='right') / len(x)
```

```
    return f

def kantil_farki(x: np.ndarray, y: np.ndarray, tau: np.ndarray) -> np.ndarray:
    """Δ(τ) = q_τ^x - q_τ^y."""
    qx = np.quantile(x, tau, method='linear')
    qy = np.quantile(y, tau, method='linear')
    return qx - qy

def _blok_bootstrap(seri: np.ndarray, blok_boy: int) -> np.ndarray:
    """Hareketli blok bootstrap ile yeniden örnekleme."""
    n = len(seri)
    n_blok = int(np.ceil(n / blok_boy))
    baslangiclar = np.random.randint(0, n - blok_boy + 1, size=n_blok)
    bloklar = [seri[b:b+blok_boy] for b in baslangiclar]
    boot = np.concatenate(bloklar)[:n]
    return boot

def kantil_farki_guven_araligi(x: np.ndarray, y: np.ndarray,
                               blok_boy: int, n_bootstrap: int = 1000,
                               alfa: float = 0.05) -> Tuple[np.ndarray, np.ndarray, np.ndarray,
np.ndarray]:
    """Kantil farkı fonksiyonu için blok bootstrap güven bandı. Ayrıca gözlenen farkı da
döndürür."""
    tau = np.linspace(0.01, 0.99, 50)
    delta_gozlenen = kantil_farki(x, y, tau)
    boot_deltalar = np.zeros((n_bootstrap, len(tau)))
    for b in range(n_bootstrap):
        x_boot = _blok_bootstrap(x, blok_boy)
        y_boot = _blok_bootstrap(y, blok_boy)
        boot_deltalar[b] = kantil_farki(x_boot, y_boot, tau)
    alt = np.percentile(boot_deltalar, 100 * alfa / 2, axis=0)
    ust = np.percentile(boot_deltalar, 100 * (1 - alfa / 2), axis=0)
```

```
    return tau, delta_gozlenen, alt, ust
def cliffs_delta(x: np.ndarray, y: np.ndarray) -> float:
    """Cliff's Delta = P(x > y) - P(x < y)."""
    n, m = len(x), len(y)
    gt = sum(np.sum(y < xi) for xi in x)
    lt = sum(np.sum(y > xi) for xi in x)
    return (gt - lt) / (n * m)
def ks_uzakligi(x: np.ndarray, y: np.ndarray) -> float:
    return ks_2samp(x, y)[0]
def wasserstein_uzakligi(x: np.ndarray, y: np.ndarray) -> float:
    return wasserstein_distance(x, y)
#
=================================================================
=========
# 🛠️DÜZELTİLMİŞ blok_permutasyon_testi (tek boyutlu dizi olarak birleştirme)
#
=================================================================
=========
def blok_permutasyon_testi(x: np.ndarray, y: np.ndarray,
                           blok_boy: int,
                           test_istatistigi: Callable[[np.ndarray, np.ndarray], float],
                           n_perm: int = 999,
                           alternatif: str = 'iki-yanli') -> Tuple[float, np.ndarray]:
    n = len(x)
    if len(y) != n:
        raise ValueError("Seriler aynı uzunlukta olmalı.")
    if n % blok_boy != 0:
        n = (n // blok_boy) * blok_boy
        x, y = x[:n], y[:n]
        warnings.warn(f"Blok boyutuna bölünebilirlik için seriler {n} uzunluğuna
```

```
kısaltıldı.")
    n_blok = n // blok_boy
    x_bloklar = x.reshape(n_blok, blok_boy)
    y_bloklar = y.reshape(n_blok, blok_boy)
    gozlenen = test_istatistigi(x, y)
    perm_istatistikler = np.zeros(n_perm)
    for i in range(n_perm):
        perm = np.random.rand(n_blok) < 0.5
        # DİKKAT: .ravel() ile düzleştirme yapıldı
        x_perm = np.concatenate([x_bloklar[perm], y_bloklar[~perm]]).ravel()
        y_perm = np.concatenate([x_bloklar[~perm], y_bloklar[perm]]).ravel()
        perm_istatistikler[i] = test_istatistigi(x_perm, y_perm)
    if alternatif == 'iki-yanli':
        p = np.mean(np.abs(perm_istatistikler) >= np.abs(gozlenen))
    elif alternatif == 'buyuk':
        p = np.mean(perm_istatistikler >= gozlenen)
    elif alternatif == 'kucuk':
        p = np.mean(perm_istatistikler <= gozlenen)
    else:
        raise ValueError("alternatif 'iki-yanli', 'buyuk' veya 'kucuk' olmalı.")
    return p, perm_istatistikler
def optimal_blok_boy(seri: np.ndarray) -> int:
    """Hall kuralı benzeri sezgisel: n^(1/3)."""
    n = len(seri)
    return int(np.floor(n ** (1/3)))
def duyarlilik_analizi(x: np.ndarray, y: np.ndarray,
                       blok_boyutlari: List[int],
                       perm_sayilari: List[int],
```

```
                    test_istatistigi: Callable) -> Dict[Tuple[int, int], float]:
    sonuc = {}
    for bb in blok_boyutlari:
        for nperm in perm_sayilari:
            p, _ = blok_permutasyon_testi(x, y, bb, test_istatistigi, nperm)
            sonuc[(bb, nperm)] = p
    return sonuc

#
==========================================================================
=========

# 📊 GÖRSELLEŞTİRME: Kantil Farkı Grafiği (düzeltilmiş)

#
==========================================================================
=========

def plot_quantile_difference(tau: np.ndarray, observed: np.ndarray,
                             lower: np.ndarray, upper: np.ndarray,
                             alpha: float = 0.05,
                             title: str = "Quantile Difference Plot",
                             xlabel: str = "Quantile (τ)", ylabel: str = "Δ(τ) = q_τ^x - q_τ^y",
                             save_path: Optional[str] = None,
                             show: bool = True):
    """
    Kantil farkı fonksiyonunu ve %(1-α) güven bandını çizer.
    """
    plt.figure(figsize=(8, 5))
    plt.fill_between(tau, lower, upper, color='gray', alpha=0.3,
                     label=f'%{(1-alpha)*100:.0f} Güven Bandı')
    plt.plot(tau, observed, 'k-', linewidth=2, label='Δ(τ) (gözlenen)')
    plt.axhline(y=0, color='r', linestyle='--', linewidth=1, label='Δ(τ)=0')
    plt.xlabel(xlabel)
```

```
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend()
    plt.grid(alpha=0.4)
    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')
        print(f"Grafik kaydedildi: {save_path}")
    if show:
        plt.show()
    else:
        plt.close()

# ==========================================================================

# 🧪ANA TEST FONKSİYONU

# ==========================================================================

def iki_orneklem_dagilim_kaymasi_testi(x: np.ndarray, y: np.ndarray,
                                       blok_boy: Optional[int] = None,
                                       n_perm: int = 999,
                                       alfa: float = 0.05,
                                       duraganlik_kontrol: bool = True,
                                       duyarlilik: bool = False,
                                       plot_quantile: bool = False,
                                       plot_title: str = "Quantile Difference with Bootstrap CI") ->
Dict[str, Any]:
    """
    Bağımlı iki örneklemde dağılım kayması testi.
    """
```

```
    n = len(x)
    if len(y) != n:
        raise ValueError("Seriler aynı uzunlukta olmalı.")
    if duraganlik_kontrol:
        dur_x = adf_duraganlik_testi(x)
        dur_y = adf_duraganlik_testi(y)
        if not dur_x:
            warnings.warn("x serisi durağan olmayabilir (ADF testi).")
        if not dur_y:
            warnings.warn("y serisi durağan olmayabilir (ADF testi).")
    else:
        dur_x = dur_y = None
    if blok_boy is None:
        blok_boy = optimal_blok_boy(x)
    if n % blok_boy != 0:
        n = (n // blok_boy) * blok_boy
        x, y = x[:n], y[:n]
        warnings.warn(f"Blok boyutuna bölünebilirlik için seriler {n} uzunluğuna
kısaltıldı.")
    ks = ks_uzakligi(x, y)
    wass = wasserstein_uzakligi(x, y)
    cliff = cliffs_delta(x, y)
    p_ks, _ = blok_permutasyon_testi(x, y, blok_boy, ks_uzakligi, n_perm)
    p_wass, _ = blok_permutasyon_testi(x, y, blok_boy, wasserstein_uzakligi, n_perm)
    tau, delta_obs, alt, ust = kantil_farki_guven_araligi(x, y, blok_boy, alfa=alfa)
    if plot_quantile:
        plot_quantile_difference(tau, delta_obs, alt, ust, alfa,
                                 title=plot_title)
```

```
cliff_boot = np.zeros(1000)
for i in range(1000):
    xb = _blok_bootstrap(x, blok_boy)
    yb = _blok_bootstrap(y, blok_boy)
    cliff_boot[i] = cliffs_delta(xb, yb)
cliff_ci = np.percentile(cliff_boot, [100*alfa/2, 100*(1-alfa/2)])
duyarlilik_sonuc = None
if duyarlilik:
    bloklar = [optimal_blok_boy(x), int(np.floor(n**(1/2))), int(np.floor(n**(2/3)))]
    permler = [500, 1000, 2000]
    duyarlilik_sonuc = duyarlilik_analizi(x, y, bloklar, permler, ks_uzakligi)
sonuc = {
    'n': n,
    'blok_boyutu': blok_boy,
    'ks_uzakligi': ks,
    'ks_p_degeri': p_ks,
    'wasserstein_uzakligi': wass,
    'wasserstein_p_degeri': p_wass,
    'cliffs_delta': cliff,
    'cliffs_delta_guven_araligi': cliff_ci,
    'kantil_farki': (tau, delta_obs, alt, ust),
    'duraganlik_x': dur_x,
    'duraganlik_y': dur_y,
    'duyarlilik': duyarlilik_sonuc,
    'h0_red_ks': p_ks < alfa,
    'h0_red_wass': p_wass < alfa,
    'uyari': (
        "Bu test aşağıdaki varsayımlara dayanır ve bunları kanıtlamaz:\n"
```

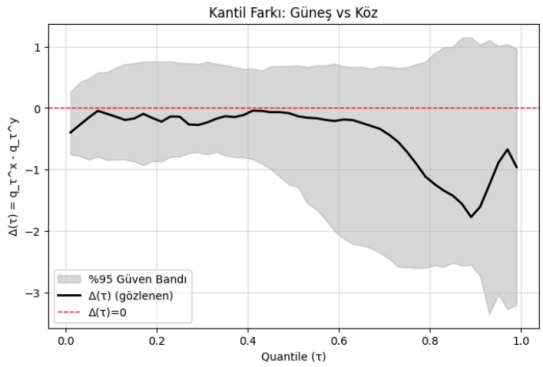
```
        "(A) Zayıf bağımlılık (α-karıştırma) sadece varsayılmıştır, ADF testi bunu doğrulamaz.\n"

        "(B) Blok permütasyon testi asimptotik olarak geçerlidir; kesin değildir, değişebilirlik yaklaşıktır.\n"

        "(C) Blok boyutu seçimi sezgisel (n^(1/3)) olup optimal değildir.\n"

        "(D) Test sadece dağılım kaymasını tespit eder, nedensellik veya değişimin nedenini açıklamaz."

      )

   }

   return sonuc

# ================================================================================

# 🧪 ÖRNEK KULLANIM: Kaya ayına göre gruplandırılmış iki serinin test edilmesi

# ================================================================================

def simule_et_ve_test_et():

   """

   Rastgele bağımlı bir süreç üretir, her gün için Kaya ayını hesaplar,

   iki farklı Kaya ayındaki gözlemleri ayırır ve dağılımların aynı olup

   olmadığını test eder. Kantil farkı grafiğini de çizer.

   """

   print("Kaya Takvimi ile Dağılım Kayması Testi Örneği")

   print("-" * 60)

   # 1. Bir yıllık zaman aralığı oluştur (2024)

   start = datetime(2024, 1, 1, 0, 0, 0, tzinfo=TR)

   end = datetime(2025, 1, 1, 0, 0, 0, tzinfo=TR)

   dates = []

   current = start

   while current < end:
```

```
    dates.append(current)
    current += timedelta(days=1)
# 2. Her gün için AR(1) bağımlı bir hata süreci üret (örnek: sıcaklık anomalisi)
np.random.seed(42)
T = len(dates)
epsilon = np.zeros(T)
phi = 0.6
for t in range(1, T):
    epsilon[t] = phi * epsilon[t-1] + np.random.normal(0, 1)
# 3. Her günün Kaya ayını hesapla
aylar = []
for dt in dates:
    kaya = gregorian_to_kaya(dt)
    aylar.append(kaya["month"])
# 4. İki farklı ay seç (örneğin "Güneş" ve "Köz")
ay1 = "Güneş"
ay2 = "Köz"
mask1 = [a == ay1 for a in aylar]
mask2 = [a == ay2 for a in aylar]
# Bu aylardaki hata değerlerini al
x = epsilon[mask1]
y = epsilon[mask2]
print(f"Seçilen aylar: {ay1} ({len(x)} gün) ve {ay2} ({len(y)} gün)")
print(f"Seri uzunlukları: x={len(x)}, y={len(y)}")
# Test için iki seriyi aynı uzunluğa getir (kısa olan kadar kes)
min_len = min(len(x), len(y))
x = x[:min_len]
y = y[:min_len]
```

```
    # 5. Dağılım kayması testini uygula (grafik çizdir)
    print("\n--- Test Sonuçları ---")
    test_sonuc = iki_orneklem_dagilim_kaymasi_testi(
        x, y,
        duyarlilik=True,
        plot_quantile=True,
        plot_title=f"Kantil Farkı: {ay1} vs {ay2}"
    )
    print(f"Kolmogorov-Smirnov p-değeri: {test_sonuc['ks_p_degeri']:.4f}")
    print(f"Wasserstein p-değeri: {test_sonuc['wasserstein_p_degeri']:.4f}")
    print(f"Cliff's Delta: {test_sonuc['cliffs_delta']:.4f}")
    print(f"H0 red (KS): {test_sonuc['h0_red_ks']}")
    print(f"H0 red (Wasserstein): {test_sonuc['h0_red_wass']}")
    print("\nUyarı:\n", test_sonuc['uyari'])
    # Duyarlılık analizi varsa yazdır
    if test_sonuc['duyarlilik']:
        print("\nDuyarlılık Analizi (blok boyutu, perm sayısı) -> p-değeri:")
        for (bb, nperm), p in test_sonuc['duyarlilik'].items():
            print(f"  Blok={bb}, perm={nperm}: p={p:.4f}")
#
==========================================================================
=========
# 🧪 ÇALIŞTIRMA
#
==========================================================================
=========
if __name__ == "__main__":
    # Önce takvim testi
    now = datetime.now(TR)
    print("Bugünün Kaya Takvimi karşılığı:")
```

```
print(get_full_date(now))

print("\n")

# Sonra dağılım kayması testi örneği (grafik çizilir)

simule_et_ve_test_et()
```

```
Bugünün Kaya Takvimi karşılığı:
Kaya Takvimi: 2 Filiz (Gün-B) 26 | 🌒 Hilal


Kaya Takvimi ile Dağılım Kayması Testi Örne
----------------------------------------
Seçilen aylar: Güneş (32 gün) ve Köz (31 gü
Seri uzunlukları: x=32, y=31

--- Test Sonuçları ---
/tmp/ipykernel_2094/429572675.py:316: UserW
  warnings.warn("x serisi durağan olmayabil
/tmp/ipykernel_2094/429572675.py:318: UserW
  warnings.warn("y serisi durağan olmayabil
/tmp/ipykernel_2094/429572675.py:327: UserW
  warnings.warn(f"Blok boyutuna bölünebilir
```




```
/tmp/ipykernel_2094/429572675.py:221: UserW
  warnings.warn(f"Blok boyutuna bölünebilir
Kolmogorov-Smirnov p-değeri: 0.1812
Wasserstein p-değeri: 0.0631
Cliff's Delta: -0.1333
H0 red (KS): False
H0 red (Wasserstein): False

Uyarı:
 Bu test aşağıdaki varsayımlara dayanır ve
(A) Zayıf bağımlılık (α-karıştırma) sadece
(B) Blok permütasyon testi asimptotik olara
(C) Blok boyutu seçimi sezgisel (n^(1/3)) o
```